Program Upload Guide

Updated Sunday, December 13, 2020

Review guidelines in www.hashimi.ws/cs324/grade.php

- Organize source code inside code starters as instructed
- Design your program output to match reference output
- Use http://jsbeautifier.org as a guide to format code (pick "Braces on own line") in 4th drop-down list

Final Project ~~Milestone 1~~ (Final Grade) – Due Midnight ~~Sunday December 13~~ Saturday December 19, 2020

1. Team tasks:
   - Task 1 (2 members): Maximum flow method to the network object based on Edmonds-Karp algorithm + any support methods (for both network and graph objects, no need for a .front method) + unit testing + JSDOC comments
   - Task 2: Network object based on Graph() + complete Vertex() methods + unit testing + JSDOC comments
   - Task 3: Output function, input files (note naming: _v, _v2, …), integration testing, upload
2. Use correct code starter (remove transitive/greedy package code both from Graph object and implementing funcs sections)
3. List your name as @author for your code in API documentation
4. Upload (Member 1 only) as: 11.js
5. Note: submit homework problem set 10.2 in the same week in your lab period.

```
<print before running max flow method call (note zero flow), edge format: u-v cap flow, … >
Network {Figure 10.4 (Levitin, 3rd edition)} DIRECTED - 6 VERTICES, 7 EDGES:
0 {1} 0-1 2 0, 0-3 3 0
1 {2} 1-2 5 0, 1-4 3 0
2 {3} 2-5 2 0
3 {4} 3-2 1 0
4 {5} 4-5 4 0
5 {6}

<print network after max flow method, must have an empty line between before-after>

Num paths: <print number of augmented paths returned by Edmonds-Karp(), see line 16 in pseudocode>
Max: <print theoretical max num of augmenting paths for family of instances>
Average path length (edges):

<repeat for the exercise>
Network {Exercise 10.2: 2b (Levitin, 3rd edition)} DIRECTED - ? VERTICES, ? EDGES:
 ...
```

Project 2 (20% Final Grade) – Due Midnight Sunday November ~~1~~ 8, 2020

Assignment checklist:

1. Team task assignment:
   - Member 1: Output function, test input files*, integration testing, API docs generation and quality assurance, upload
   - Member 2: Second Prim's implementation + unit testing + JSDOC comments
   - Member 3: Dijkstra's algorithm + unit testing + JSDOC comments
   - Member 4: First priority queue + unit testing + JSDOC comments
2. Paste priority queue implementing functions only at the very end (do not paste pq objects)
3. List your name as author for your code in API as discussed in class

4. Upload (Member 1 only): 101.js
5. Note: submit individually homework problem set 9.3 in your lab period.

```
GRAPH {Exercise 9.2: 1b (Levitin, 3rd edition)} WEIGHTED, UNDIRECTED - 12 VERTICES, 40 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1,2,3
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 0,4,5
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,6
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2,4,7
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 1,3,5,8
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 1,4,9
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 2,7,10
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 3,6,10,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 4,7,9,11
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 5,8,11
VERTEX: 10 {k} - VISIT: false - ADJACENCY: 6,7,11
VERTEX: 11 {l} - VISIT: false - ADJACENCY: 8,9,10

dfs_push: 0,1,4,3,2,6,7,10,11,8,9,5

CONNECTED

bfs_order: 0,1,2,3,4,5,6,7,8,9,10,11

Distance matrix (Warshall-Floyd)
<show all rows like P1M2>

MST by Prim2 (linear PQ)
<Sequence of edges from an enhanced VT output array, where source vertex is in 'parent' field and target
vertex (VT) is in 'tree'. Example (Figure 9.3): (-,0), (0,1), (1,2), (1,5), (5,4), (5,3). Note the standard
(non-optional) output array field names.>

Shortest paths by Dijkstra from vertex 0
<like MST, using same output array, store and show path lengths from 3rd optional 'distance' field>
Total path distance comps (step 11):
Total PQ updates (step 13):

Distance matrix from Dijkstra
<repeat to check against result above from Warshall-Floyd>
```

* Test many more than Exercise 9.2-1b, observe efficiency parameters (path distance comps, PQ updates).

Programming Checkpoint 10 – Due with Project 2

Assignment checklist (Member 4)

1. Meet team to discuss design of PQ package based on starter comments and design notes from the course group under Project 2 Discussion topic (see reference link in assignment page)
2. Document design after agreement with team in the PQ starter (no programming)
3. Document encapsulation violation in pq.js (linklist.js should not be touched)
4. ~~Generate API docs and share with team (help in assignment page)~~
5. Upload the completed design file (no code) as: 10.js

Project 1 Milestone 2 (25% Final Grade) – Due Midnight Sunday October 18, 2020

Assignment checklist:

1. Complete first graph implementation according to published specs (except *makeGraph*)
2. Test your TC methods using various types of graphs to ensure proper coding
3. Implement Prim's algorithm as discussed in class, include suitable JSDOC comments, test using 12-city dataset from assignment page
4. Use input array names: _v[]/_e[], _v2[]/_e2[], and _v3[]/_e3[], respectively, for input files: exercise8_4_1.js, exercise8_4_7.js, and ksa12city.js (note global vars naming)
5. Upload name: 9.js

```
GRAPH {Exercise 8.4: 1 (Levitin, 3rd edition)} DIRECTED - 4 VERTICES, 3 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 2
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 3
VERTEX: 3 {d} - VISIT: false - ADJACENCY:

dfs_push: 0,1,2,3

CONNECTED

TC matrix by DFS:
<similar to Checkpoint 8>

TC matrix by Warshall-Floyd:
<similar to Checkpoint 8>

DAG: <return value of .isDAG()>

TC matrix by Warshall-Floyd Exercise 8.4: 7
<similar to Checkpoint 8>

Distance matrix Exercise 8.4: 7
<similar to Checkpoint 8>

DAG Exercise 8.4: 7
<return value of .isDAG()>

MST for KSA 12 cities
<your way; surprise me!>
```

Programming Checkpoint 8 – Due Midnight Sunday October 11, 2020

Assignment checklist:

1. Complete transitive closure package (details in code starter 8) – do your best
2. Include JSDOC comments for your new TC package methods (only)
3. Use graph input file name: exercise8_4_7.js (compare results in course group, discuss difficulties)
4. Upload name: 8.js
5. Note: submit homework problem set 8.4 in the same week in your lab period

```
GRAPH {Exercise 8.4: 7 (Levitin, 3rd edition)} WEIGHTED, DIRECTED - 5 VERTICES, 10 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1,3,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 0,2,3
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 3
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 2,4
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 0

bfs_order: 0,1,3,4,2

CONNECTED

TC matrix by DFS:
<similar format to previous checkpoint>

TC matrix by Warshall-Floyd:
<should match above>

DAG: <return value of .isDAG()>
```

```
Distance matrix
<similar to TC but show distances of course>
```

[INTERNAL] Programming Checkpoint – Due …

Assignment checklist:

1. Use the correct starter (8)
2. Add a dfs-based transitive closure method to Graph
3. Use the JSDOC demo as example to document the new dfs-based transitive closure method (only)
4. Create graph input files for figures 8.11, 8.14 (like figure3_10.js) and name similarly
5. Use to test* your code and to compare results

```
GRAPH {Figure 8.11 (Levitin, 3rd edition)} DIRECTED - 4 VERTICES, 4 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 3
VERTEX: 2 {c} - VISIT: false - ADJACENCY:
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2

dfs_push: 0,1,3,2

<print connectivity status>

bfs_order: 0,1,3,2

TC matrix:
1,1,1,1
1,1,1,1
0,0,0,0
1,1,1,1
```

* Verify that a zero-row signifies (means) a graph is disconnected. Can the number of connected components be found from the TC matrix?

Project 1 Milestone 1 (15% Final Grade) – Due Midnight Friday October 2, 2020

Assignment checklist:

1. Review carefully the published documentation of the first graph implementation, and apply the following changes as discussed in class:
   - Update object properties to match
   - Update names of methods and their implementing functions to match
   - Implement connectivity and vertex interfaces
   - Fix encapsulation issue in `.makeAdjMatrix` and `.addEdge` implementation as described by specs (keep older impl, name new ones: <method name>Impl2, Impl3, ...)
2. Ensure output from Checkpoint 6 is still correct for both weighted and unweighted inputs.
3. Prepare your code as described at the top (ensure source based on starter 6)
4. Upload name: 7.js

Programming Checkpoint 6 – Due Midnight Sunday September 27, 2020

Assignment checklist:

1. Complete code necessary to support weighted graphs through an optional edge weight
2. Modify the adjacency matrix generator method to use weights if graph is weighted (instead of 1s)
3. Create a version of the input file (call it figure3_10w.js) where an optional `.w` field specifies an edge weight, use value range 10-21 (example: `{u:0, v:3, w:10}` for first input edge)
4. Test with no optional weight to ensure that you still get the results of previous internal checkpoint (note change in `.print_graph()` behavior)
5. Upload name: 6.js

```
GRAPH {Figure 3.10 (Levitin, 3rd edition)} WEIGHTED, UNDIRECTED - 10 VERTICES, 24 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6

dfs_push: 0,3,2,5,4,1,6,7,8,9

DISCONNECTED (2)

bfs_order: 0,3,2,4,5,1,6,7,9,8

first row matrix: 0,0,11,10,17,0,0,0,0,0

last row matrix: 0,0,0,0,0,0,21,0,20,0
```

[INTERNAL] Programming Checkpoint – ~~Due~~ …

Assignment checklist:

1. Use starter 6 (commit using message: Internal checkpoint)
2. Add an edge object with 1 property field: `target_v`
3. Change `.adjacentByID()` method to extract target id from new edge object
4. Add `.connectedComp` property field to Graph as follows: initially 0 (for no information), otherwise number of connected components
5. Merge BFS and DFS methods into a master topological search method `.topoSearch()`

6. Add code in `.topoSearch()` to set `.connectedComp` correctly
7. Add a method to generate an adjacency matrix (see the 2-dim array language demo)

```
GRAPH {Figure 3.10 (Levitin, 3rd edition)} UNDIRECTED - 10 VERTICES, 24 EDGES:

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6

no connectivity info

dfs_push: 0,3,2,5,4,1,6,7,8,9

DISCONNECTED (2)

bfs_out: 0,3,2,4,5,1,6,7,9,8

first row matrix: 0,0,1,1,1,0,0,0,0,0

last row matrix: 0,0,0,0,0,0,1,0,1,0
```

Programming Checkpoint 5 – Due Midnight Tuesday Thursday September 22, 2020

Assignment checklist:

1. Code a `.deleteFirst()` method to List, reuse to implement `.dequeue()`
2. Add an `.adjacentByID()` method to Vertex to replace explicit calls to `.traverse()` in Graph
3. Add a `.label` property to Graph, re-implement `.print_graph()`
4. Reorganize main script file as discussed in class (use starter 5 and the new caller page)
5. Link a graph input file (name: figure3_10.js), check output
6. Upload name: 5.js

```
GRAPH {Figure 3.10 (Levitin, 3rd edition)} UNDIRECTED - 10 VERTICES, 24 EDGES:

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6

0,3,2,5,4,1,6,7,8,9

0,3,2,4,5,1,6,7,9,8
```

Programming Checkpoint 4 – Due Midnight Sunday September 20, 2020

Assignment checklist:

1. Implement the basic BFS (use starter 4 as main script file)
2. Complete queue.js as explained in class (use the starter)
3. Test main script file after linking to queue.js
4. Upload (main script file) as: 4.js

```
0,3,2,5,4,1,6,7,8,9

0,3,2,4,5,1,6,7,9,8
```

Programming Checkpoint 3 – Due Midnight Tuesday September 15, 2020

Assignment checklist:

1. Prepare code as described above (use starter 3, watch standardized field names)
2. Upload name: 3.js

```
VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6

0,3,2,5,4,1,6,7,8,9
```

Bonus Points ✪ Get pop order of depth-first search (store as property of Graph() similar to push order)

Programming Checkpoint 2 – Due Midnight Sunday September 13, 2020

Assignment checklist:

1. Review the linked-list usage demo (focus interfaces)
2. Use starter 2
3. Prepare your code as described above
4. Upload name: 2.js

```
VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6
```

Programming Checkpoint 1 – Due Midnight Tuesday September 8, 2020

Assignment checklist:

1. Study the language demo
2. Use starter code as your main script file
3. Read comments carefully and follow instructions in the starter
4. Prepare your code as described above*
5. Upload main script file as: 1.js

```
VERTEX: 0 {jeddah} - VISIT: false - ADJACENCY: null
VERTEX: 1 {makkah} - VISIT: false - ADJACENCY: null
VERTEX: 2 {madinah} - VISIT: false - ADJACENCY: null
```

*Note: your output should show your labels